0325.00487
CD01060

TECHNIQUES FOR JEDEC FILE INFORMATION INTEGRITY AND PRESERVATION OF DEVICE PROGRAMMING SPECIFICATIONS

Field of the Invention

The present invention relates to a method and/or architecture for file integrity generally and, more particularly, to JEDEC file integrity and preservation of programming specifications.

Background of the Invention

Files generated in compliance with the Joint Electron Device Engineering Council (JEDEC) standard JESD3-C are used for programming field programmable logic devices. The JESD3-C standard defines a format for transferring fuse or cell states between a development system and a programmer. The files consist of (i) non-programming fields within a design specification portion, (ii) device programming fields, and (iii) a cyclic redundancy check (CRC) portion.

Programming a device starts with presenting parameters defining a program to the development system along with non-

programming type information. The development system generates programmable items from the parameters. The programmable items are then stored in the programmable fields of the file. The non-programmable type information is stored in the non-programming fields of the file.

The JESD3-C standard has several practical limitations. One limitation is that the files cannot be used to reverse calculate the original parameters. A common practice is to dispose of the original device programming parameters once the file has been verified as correct and the programmed devices have gone into production. Consequently, the original intent of the program is lost with the discarding of the parameters.

Summary of the Invention

The present invention concerns a method of generating a file suitable for programming a programmable logic device. The method generally comprises the steps of (A) generating a programming item from a plurality of parameters that define a program for the programmable logic device, (B) storing the programming item in a programming field of the file in response to

generating, and (C) storing at least one of the parameters in a non-programming field of the file.

The objects, features and advantages of the present invention include providing a method and/or architecture for enhanced JEDEC file generation that may (i) preserve the ability to fully recreate the original parameters from the file, (ii) detect a change in bit position/bit values, (iii) substantially reduce tracking through file validation, (iv) preserve input and output frequency parameters, and/or (v) provide full backward compatibility to pre-existing and future JEDEC file readers.

Brief Description of the Drawings

These and other objects, features and advantages of the present invention will be apparent from the following detailed description and the appended claims and drawings in which:

FIG. 1 is a block diagram of a system implementing the present invention;

FIG. 2 is an exemplary file created by the present invention; and

FIG. 3 is a flow diagram of a process for generating and using a modified JEDEC file.

Detailed Description of the Preferred Embodiments

Referring to FIG. 1, a block diagram of a system 100 is shown in accordance with a preferred embodiment of the present invention. The system 100 may be configured to program a programmable logic device 102 based upon one or more input items 104. The system 100 generally comprises a circuit 106, a transfer medium 108, a circuit 110, and a circuit 112. Other embodiments of the system 100 may include only one of the circuit 110 or the circuit 112. The circuit 106 may be implemented as a development system. The circuit 110 may be implemented as a hardware programmer. The circuit 112 may be implemented as a generic file reader and/or parsing engine. The transmission medium 108 may connect the development system 106 to the hardware programmer 110 and/or the file reader/parsing engine 112 to transfer a file 114.

The development system 106 may be configured to generate the file 114 from the input items 104. The input items 104 may include one or more programming parameters 116. The programming parameters 116 may define a program for configuring the programmable logic device 102. The input items 104 may optionally include one or more information items 118. The information items 118 may be non-programmable data stored in the file 114 for

reference purposes. The information items 118 generally do not affect programming of the programmable logic device 102.

The development system 106 may comprise a processor 120 and software 122 residing in a memory 124. The processor 120 may be configured to execute the software 122. The software 122 may instruct the processor 120 to generate the file 114 from the input items 104. The processor 120 may format the file 114 per the Joint Electron Device Engineering Council (JEDEC) standard JESD3-C, hereby incorporated by reference in its entirety. The JESD3-C standard is published by the Electronic Industries Association, Washington, D.C.

The hardware programmer 110 may be controlled by software 126 to program the programmable logic device 102. The programmable logic device 102 may be connected to the hardware programmer 110 through a socket 128 or similar device. The hardware programmer 110 may receive the file 114 from the development system 106 through the transfer medium 108. The hardware programmer 110 may then program or flash the programmable logic device 102 using the programmable items stored within the file 114.

The transfer medium 108 may be configured to distribute the file 114 among the development system 106, the hardware

programmer 110, and the file reader/parsing engine 112. The transfer medium 108 may be implemented as a serial communication link, a network, a removable medium such as a disk or tape, or the like. Other forms of the transfer medium 108 may be implemented to meet the design criteria of a particular application.

The file reader/parsing engine 112 may be configured to parse the file 114. The parsing may recover the programming parameters 116 and the information items 118 saved as part of the file 114. The recovered items may be useful in determining the original intent of the program defined by the programming parameters 116. The recovered items may also be useful in debugging and/or analyzing the program defined therein.

Referring to FIG. 2, a diagram of an example file 114 implementing the present invention is shown. The file 114 generally adheres to a specific protocol that instructs the hardware programmer 110 on three basic principles. A first section 130 may be a comment section that outlines device identification and other comments. The fields within the first section 130 may be non-programming. A second section 132 may contain programming fields that store addressing information about where to start placing bits. The event of programming generally begins at a first

address location and proceeds in a linear fashion until either an end-of-file is reached or a new address is detected. The second section 132 may be passed through a simple cyclic redundancy check (CRC) process and compared against a CRC value stored in a final section 134. The comparison may verify the accuracy of the contents of the file 114.

One or more special comment lines 136 may be included in the first section 130 to store some or all of the programming parameters 116 and the information items 118. In general, the comment lines 136 may comprise one or more lines and/or fields bracketed by delimiters. Existing and future third-party hardware programmers 110, file readers/paring engines 112, and other equipment that may read the file 114 may be compatible the comment lines 136 per the JESDC-3 standard because the comment lines 136 are within the first section 130.

A general format of a first comment line 136 may be given by, "x Checksum: aaaaaaaa". The field "x" may represent a starting delimiter (e.g., "<"). The field "aaaaaaaa" may be implemented as a non-programming field than may store an error detection item. In one embodiment, the error detection item may be a unique CRC checksum that may be independent from the JESD3-C standard CRC

value. The unique CRC checksum may be used by the development systems 106, the hardware programmers 110, the file readers/parsing engines 112, and other equipment to verify the file 114.

A general format of a second comment line 136 may given by "# z bbbbbbbbbb". The field "z" may represent another type of delimiter (e.g., "s"). The field "bbbbbbbbbb" may be implemented as a non-programming field that may store an information item 118. In one embodiment, the delimiter "s" may indicate that the following field "bbbbbbbbbb" may contain a device part number.

A general format of additional comment lines 136 may be given by "# w ccccccccc ; ddddddddd". The field "w" may represent another delimiter (e.g., "f"). The field "ccccccccc" may be implemented as a non-programming field that may store a programming parameter 116. The field "ddddddddd" may be implemented as a non-programming field that may store an information item 118. In one embodiment, the delimiter "f" may indicate that the following field "ccccccccc" may contain a frequency used to program the programmable logic device 102. The field "ddddddddd" may contain a comment describing the programming parameter 116 stored in the field "ccccccccc".

A general format of a final comment line 136 may be given by "# w ccccccccc ; ddddddddd y". The field "y" may represent an ending delimiter (e.g., ">"). Additional fields and delimiters may be defined within the comment lines 136 to meet the design criteria of a particular application.

The delimiters may be implemented as a character, a symbol, a number, or the like. The delimiters may distinguish the comment lines 136 to the development systems 106, the hardware programmers 110, the file readers/parsing engines 112, and other equipment capable of reading the file 114. In one embodiment, the presence of carriage returns and line feeds between the delimiters may be ignored.

In an example, the programmable logic device 102 may be a programmable multiple phase-lock loop (PLL) device. The following applications may be used for the various fields of the comment lines 136. The "aaaaaaaa" field may store the unique checksum value (e.g., "74F47101") for the file 114. The "bbbbbbbbbb" field may store the device part number (e.g., "NMN3344"). The following fields "ccccccccc" and "ddddddddd" may store frequencies at various interfaces of the programmable logic device 102. In particular, the third comment line 136 may store an

external reference frequency. The fourth through last comment lines 136 may store output frequencies for the various PLLs.

Generation of the unique CRC checksum may consist of a series of registers. Starting immediately after the unique CRC checksum field (e.g., the "aaaaaaaa" field) or starting from the second portion 132, the file 114 may be read as an input data stream into a first register sequentially and "clocked" through the series of registers in tandem. Feedback may be provided where values of some registers are feed back to some previous registers. The feedback may be dependent upon coefficients that define a polynomial. Presentation of the input data stream into the generation process may continue until the whole ASCII "binary" bit input data stream has been entered. Once the entire input data stream has been entered, taps from specific registers may be used to generate the unique CRC checksum value.

The accuracy of the generation process generally depends on the number of registers and the coefficients of the polynomial. In one embodiment, there may be 21 stage registers and a predetermined polynomial. The combination of 21 registers and the predetermined polynomial may achieve a detection accuracy to a resolution that may detect a presence of a bit swap(s). Bit

swapping may include swapping of position and/or swapping of values. Other combinations of registers and coefficients may be implemented to meet the design criteria of a particular application.

The generation process may be implemented in hardware, firmware, software, micro code, or any combination of hardware, firmware, software and micro code. The following is an example of a unique CRC checksum generation process implemented as a software program having 21 registers and a polynomial having all initial coefficients set to zero. The input data stream may be represented by an array[s]. The example code may be:

```
void   SONOS::calculateChecksum(int tempEnd, char array[], char
checksum[])
{
    unsigned short reg[21];
    unsigned short temp;

    int j = 20;

    for (int k = 0; k < 21; k++)
    {
```

```
            reg[k] = 0;
        }
        temp = reg[20];

    // There are 21 registers to store polynomial coefficients
    // and 7 adders to calculate unique CRC checksum in this process
        for (int s = 0; s < tempEnd; s++)
        {
            reg[0] = array[s] + temp;
            reg[3] = reg[2] + temp;
            reg[6] = reg[5] + temp;
            reg[9] = reg[8] + temp;
            reg[12] = reg[11] + temp;
            reg[15] = reg[14] + temp;
            reg[18] = reg[17] + temp;

            temp = reg[20];

            // Shifting registers
            for (int j = 20; j > 0; j--)
            {
```

```
            reg[j] = reg[j-1];
        }
    }
    int tempCurrent = 0;
      tempCurrent += sprintf (checksum + tempCurrent, "%4X%4X",
reg[15], reg[10]);
    checksum[tempCurrent] = '\0';
}
```

Other unique CRC checksum generation processes may be implemented to meet the design criteria of a particular application.

Referring to FIG. 3, a flow diagram of a method of generating and using the file 114 is shown. The process may begin with reception of the programming parameters 116 and the information items 118 by the development system 106 (e.g., block 138). The processor 120 may then generate the programming items from the programming parameters 116 (e.g., block 140). The processor 120 may store, in no particular order, (i) the programming items in the programmable fields, (ii) selected programming parameters 116 in the non-programming fields of the comment lines 136, (iii) the identification item in a non-programming field of the comment lines 136, and (iv) add the

starting delimiter and the ending delimiter to the comment lines 136 (e.g., block 142).

Once the processor 120 has generated the second portion 132 of the file 114, the processor 120 may generate the unique CRC checksum (e.g., block 144). The unique CRC checksum may be stored in a non-programmable field of the comment lines 136 (e.g., block 146). Optionally, the completed file 114 may be stored within the development system 106 for later transmission (e.g., block 148).

To program a programmable logic device 102, the file 114 may be transferred from the development system 106 to the hardware programmer 110 (e.g., block 150). The hardware programmer 110 may then use the second portion 132 of the file 114 to flash the program into the programmable logic device 102 (e.g., block 152). Programming may then be repeated with additional programmable logic devices 102 using the same file 114.

To recover the saved programming parameters 116, identification items 118, and/or the unique CRC checksum, the file 114 may be transferred to the file reader/parsing engine 112 (e.g., block 150). The file reader/parsing engine 112 may parse the file 114 using the delimiters to extract the comment line or lines 136 (e.g., block 156). The comment lines 136 may then be separated by

the fields into the identification items 118, the programming parameters 116 and the unique CRC checksum (e.g., block 156).

The file 114 may then be validated against the unique CRC checksum (e.g., block 158). If the file 114 passes the validation (e.g., the YES branch of decision block 160), then the extracted programming parameters 116 and identification items 118 may be presented for display (e.g., block 162). In situations where the file reader/parsing engine 112 also has a programming capability, then the validated second portion 132 of the file 114 may be used to program the programmable logic device (e.g., block 164). If the file 114 fails the validation (e.g., the NO branch of decision block 160), then an error message may be presented for display (e.g., block 166).

The function performed by the flow diagram of FIG. 3 may be implemented using a conventional general purpose digital computer programmed according to the teachings of the present specification, as will be apparent to those skilled in the relevant art(s). Appropriate software coding can readily be prepared by skilled programmers based on the teachings of the present disclosure, as will also be apparent to those skilled in the relevant art(s).

The present invention may also be implemented by the preparation of ASICs, FPGAs, or by interconnecting an appropriate network of conventional component circuits, as is described herein, modifications of which will be readily apparent to those skilled in the art(s).

The present invention thus may also include a computer product which may be a storage medium including instructions (or computer program) which can be used to program a computer to perform a process in accordance with the present invention. The storage medium can include, but is not limited to, any type of disk including floppy disk, optical disk, CD-ROM, and magneto-optical disks, ROMs, RAMs, EPROMs, EEPROMs, Flash memory, magnetic or optical cards, or any type of media suitable for storing electronic instructions.

While the invention has been particularly shown and described with reference to the preferred embodiments thereof, it will be understood by those skilled in the art that various changes in form and details may be made without departing from the spirit and scope of the invention.